Foundations of Preference Queries

Jan Chomicki
University at Buffalo
Plan of the course

1. Preference relations
2. Preference queries
3. Preference management
4. Advanced topics
Part I

Preference relations
Outline of Part I

1. Preference relations
   - Preference
   - Equivalence
   - Preference specification
   - Combining preferences
   - Skylines
Preference relations

Universe of objects

- constants: uninterpreted, numbers,
- individuals (entities)
- tuples
- sets

Preference relation \( \preceq \) is a binary relation between objects \( x \succ y \equiv x \) is better than \( y \equiv x \) dominates \( y \). Preference relations used in queries.
Preference relations

**Universe of objects**
- constants: uninterpreted, numbers, ...
- individuals (entities)
- tuples
- sets

**Preference relation ⊳**
- binary relation between objects
- \( x \succ y \equiv x \text{ is\_better\_than } y \equiv x \text{ dominates } y \)
- an abstract, uniform way of talking about desirability, worth, cost, timeliness, ..., and their combinations
- preference relations used in queries
Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Salesman: Those are the best cars, according to your preferences, that we have in stock.
Customer: Wait...it better be a BMW.
Salesman: What kind of car do you prefer?
Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Buying a car

Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Salesman: Those are the best cars, according to your preferences, that we have in stock.
Salesman: What kind of car do you prefer?
Customer: The newer the better, if it is the same make. And cheap, too.
Salesman: Which is more important for you: the age or the price?
Customer: The age, definitely.
Salesman: Those are the best cars, according to your preferences, that we have in stock.
Customer: Wait...it better be a BMW.
Properties of preference relations

- **Irreflexivity**: \( \forall x. x \not\succ x \)
- **Asymmetry**: \( \forall x, y. x \succ y \Rightarrow y \not\succ x \)
- **Transitivity**: \( \forall x, y, z. (x \succ y \land y \succ z) \Rightarrow x \succ z \)
- **Negative Transitivity**: \( \forall x, y, z. (x \not\succ y \land y \not\succ z) \Rightarrow x \not\succ z \)
- **Connectivity**: \( \forall x, y. x \succ y \lor y \succ x \lor x = y \)

Orders
- **Strict Partial Order (SPO)**: irreflexive and transitive
- **Weak Order (WO)**: negatively transitive SPO
- **Total Order**: connected SPO
Properties of preference relations

Properties of $\succ$

- **irreflexivity**: $\forall x. \ x \not\succ x$
- **asymmetry**: $\forall x, y. \ x \succ y \Rightarrow y \not\succ x$
- **transitivity**: $\forall x, y, z. \ (x \succ y \land y \succ z) \Rightarrow x \succ z$
- **negative transitivity**: $\forall x, y, z. \ (x \not\succ y \land y \not\succ z) \Rightarrow x \not\succ z$
- **connectivity**: $\forall x, y. \ x \succ y \lor y \succ x \lor x = y$

Orders

- **strict partial order (SPO)**: irreflexive and transitive
- **weak order (WO)**: negatively transitive SPO
- **total order**: connected SPO
Properties of preference relations

Properties of $\succ$

- **irreflexivity**: $\forall x. x \not\succ x$
- **asymmetry**: $\forall x, y. x \succ y \Rightarrow y \not\succ x$
- **transitivity**: $\forall x, y, z. (x \succ y \land y \succ z) \Rightarrow x \succ z$
- **negative transitivity**: $\forall x, y, z. (x \not\succ y \land y \not\succ z) \Rightarrow x \not\succ z$
- **connectivity**: $\forall x, y. x \succ y \lor y \succ x \lor x = y$

Orders

- **strict partial order (SPO)**: irreflexive and transitive
- **weak order (WO)**: negatively transitive SPO
- **total order**: connected SPO
Weak and total orders

**Weak order**

- a
- b
- c
- d
- e
- f

**Total order**

- a
- b
- c
- d
- e
- f
Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity: captures rationality of preference but not always guaranteed: voting paradoxes help with preference querying.

Negative transitivity: scoring functions represent weak orders.

We assume that preference relations are SPOs.
Irreflexivity, asymmetry: uncontroversial.
Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:
- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying
Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:
- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying

Negative transitivity:
- scoring functions represent weak orders
Order properties of preference relations

Irreflexivity, asymmetry: uncontroversial.

Transitivity:
- captures rationality of preference
- not always guaranteed: voting paradoxes
- helps with preference querying

Negative transitivity:
- scoring functions represent weak orders

We assume that preference relations are SPOs.
When are two objects equivalent?

Relation

\[ x \sim y \equiv x' \sim y' \]

Several notions of equivalence:

Equality:

\[ x \sim_{eq} y \equiv x = y \]

Indifference:

\[ x \sim_{i} y \equiv x \not\succ y \land y \not\succ x \]

Restricted indifference:

\[ x \sim_{r} y \equiv \forall z. (x \prec z \iff y \prec z) \land (z \prec y \iff z \prec x) \]

Properties of equivalence:

Equivalence relation: reflexive, symmetric, transitive.

Equality and restricted indifference (if \( \succ \) is an SPO) are equivalence relations.

Indifference is reflexive and symmetric; transitive for WO.
When are two objects equivalent?

Relation ~

- **binary** relation between objects
- \( x \sim y \equiv x \ "is\ equivalent\ to" \ y \)
When are two objects equivalent?

Relation $\sim$
- binary relation between objects
- $x \sim y \equiv x \text{ "is equivalent to" } y$

Several notions of equivalence
- equality: $x \sim^{eq} y \equiv x = y$
- indifference: $x \sim^{i} y \equiv x \not\succ y \land y \not\succ x$
- restricted indifference:
  $x \sim^{r} y \equiv \forall z. (x \prec z \iff y \prec z) \land (z \prec y \iff z \prec x)$
When are two objects equivalent?

Relation $\sim$
- binary relation between objects
- $x \sim y \equiv x \ "is \ equivalent \ to" \ y$

Several notions of equivalence
- equality: $x \sim^{eq} y \equiv x = y$
- indifference: $x \sim^{i} y \equiv x \nless y \land y \nless x$
- restricted indifference:
  $x \sim^{r} y \equiv \forall z. (x \less z \iff y \less z) \land (z \less y \iff z \less x)$

Properties of equivalence
- equivalence relation: reflexive, symmetric, transitive
- equality and restricted indifference (if $\succ$ is an SPO) are equivalence relations
- indifference is reflexive and symmetric; transitive for WO
Example

This is a strict partial order which is not a weak order.

Preference:
- $\text{bmw} \succ \text{ford}$
- $\text{bmw} \succ \text{vw}$
- $\text{bmw} \succ \text{mazda}$
- $\text{bmw} \succ \text{kia}$

Indifference:
- $\text{ford} \sim \text{vw}$
- $\text{vw} \sim \text{ford}$
- $\text{ford} \sim \text{mazda}$
- $\text{mazda} \sim \text{ford}$
- $\text{vw} \sim \text{mazda}$
- $\text{mazda} \sim \text{vw}$
- $\text{ford} \sim \text{kia}$
- $\text{kia} \sim \text{ford}$
- $\text{vw} \sim \text{kia}$
- $\text{kia} \sim \text{vw}$

Restricted indifference:
- $\text{ford} \sim_{r} \text{vw}$
- $\text{vw} \sim_{r} \text{ford}$
This is a strict partial order which is not a weak order.

Preference:
- \( bmw \succ ford, bmw \succ vw \)
- \( bmw \succ mazda, bmw \succ kia \)

Indifference:
- \( ford \sim vw, vw \sim ford, ford \sim mazda, mazda \sim ford, vw \sim mazda, mazda \sim vw, ford \sim kia, kia \sim ford, vw \sim kia, kia \sim vw \)

Restricted indifference:
- \( ford \sim_r vw, vw \sim_r ford \)
Preference:

\[ \text{bmw} \succ \text{ford}, \text{bmw} \succ \text{vw} \]
\[ \text{bmw} \succ \text{mazda}, \text{bmw} \succ \text{kia} \]
\[ \text{mazda} \succ \text{kia} \]

Indifference:

\[ \text{ford} \sim^i \text{vw}, \text{vw} \sim^i \text{ford} \]
\[ \text{ford} \sim^i \text{mazda}, \text{mazda} \sim^i \text{ford} \]
\[ \text{vw} \sim^i \text{mazda}, \text{mazda} \sim^i \text{vw} \]
\[ \text{ford} \sim^i \text{kia}, \text{kia} \sim^i \text{ford} \]
\[ \text{vw} \sim^i \text{kia}, \text{kia} \sim^i \text{vw} \]

Restricted indifference:

\[ \text{ford} \sim^r \text{vw}, \text{vw} \sim^r \text{ford} \]
Example

Preference:

bmw ≻ ford, bmw ≻ vw
bmw ≻ mazda, bmw ≻ kia
mazda ≻ kia

Indifference:

ford ∼i vw, vw ∼i ford,
ford ∼i mazda, mazda ∼i ford,
vw ∼i mazda, mazda ∼i vw,
ford ∼i kia, kia ∼i ford,
vw ∼i kia, kia ∼i vw

Restricted indifference:

ford ∼r vw, vw ∼r ford

This is a strict partial order which is not a weak order.
Not every SPO is a WO

**Canonical example**

\[
mazda \succ kia, mazda \sim^i vw, kia \sim^i vw
\]

**Violation of negative transitivity**

\[
mazda \not\succ vw, vw \not\succ kia, mazda \succ kia
\]
Preference specification

Explicit preference relations
Finite sets of pairs: $\text{bmw} \succ \text{mazda}$, $\text{mazda} \succ \text{kia}$,...

Implicit preference relations
can be infinite but finitely representable
defined using logic formulas in some constraint theory:
$((m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2)) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)$ for relation $\text{Car}(\text{Make}, \text{Year}, \text{Price})$.

defined using preference constructors (Preference SQL)
defined using real-valued scoring functions:
$F(m, y, p) = \alpha \cdot y + \beta \cdot p$
$((m_1, y_1, p_1) \succ_2 (m_2, y_2, p_2)) \equiv F(m_1, y_1, p_1) > F(m_2, y_2, p_2)$
Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,...
Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia,…

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

\[(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)\]

for relation Car(Make, Year, Price) defined using preference constructors (Preference SQL) defined using real-valued scoring functions:

\[F(m, y, p) = \alpha \cdot y + \beta \cdot p\]
Preference specification

Explicit preference relations

Finite sets of pairs: bmw ≻ mazda, mazda ≻ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

\[(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)\]

for relation \(Car(Make, Year, Price)\).
Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia, ... 

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)$$

for relation $Car(Make, Year, Price)$.
- defined using preference constructors (Preference SQL)
Preference specification

Explicit preference relations

Finite sets of pairs: bmw ≻ mazda, mazda ≻ kia,...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

\[(m_1, y_1, p_1) ≻_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)\]

for relation Car(Make, Year, Price).

- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions:
Preference specification

Explicit preference relations

Finite sets of pairs: bmw ≻ mazda, mazda ≻ kia,…

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:
  \[(m_1, y_1, p_1) ≻_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)\]
  for relation \(Car(Make, Year, Price)\).
- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions: \(F(m, y, p) = \alpha \cdot y + \beta \cdot p\)
Preference specification

Explicit preference relations

Finite sets of pairs: bmw $\succ$ mazda, mazda $\succ$ kia, ...

Implicit preference relations

- can be infinite but finitely representable
- defined using logic formulas in some constraint theory:

$$(m_1, y_1, p_1) \succ_1 (m_2, y_2, p_2) \equiv y_1 > y_2 \lor (y_1 = y_2 \land p_1 < p_2)$$

for relation Car(Make, Year, Price).
- defined using preference constructors (Preference SQL)
- defined using real-valued scoring functions: 

$$F(m, y, p) = \alpha \cdot y + \beta \cdot p$$

$$(m_1, y_1, p_1) \succ_2 (m_2, y_2, p_2) \equiv F(m_1, y_1, p_1) > F(m_2, y_2, p_2)$$
Logic formulas
Logic formulas

The language of logic formulas

- constants
- object (tuple) attributes
- comparison operators: \(=, \neq, <, >, \ldots\)
- arithmetic operators: \(+, \cdot, \ldots\)
- Boolean connectives: \(\neg, \land, \lor\)
- quantifiers:
  - \(\forall, \exists\)
  - usually can be eliminated (quantifier elimination)
Representability

Definition

A scoring function $f$ represents a preference relation $\succ$ if for all $x, y$:

$x \succ y \equiv f(x) > f(y)$.

Necessary condition for representability

The preference relation $\succ$ is a weak order.

Sufficient condition for representability

$\succ$ is a weak order if the domain is countable or some continuity conditions are satisfied (studied in decision theory).
Definition

A scoring function $f$ represents a preference relation $\succ$ if for all $x, y$

$$x \succ y \equiv f(x) > f(y).$$

Necessary condition for representability

The preference relation $\succ$ is a weak order.

Sufficient condition for representability

$\succ$ is a weak order if the domain is countable or some continuity conditions are satisfied (studied in decision theory).
Representability

**Definition**
A scoring function \( f \) represents a preference relation \( \succ \) if for all \( x, y \)

\[
x \succ y \equiv f(x) \geq f(y).
\]

**Necessary condition for representability**
The preference relation \( \succ \) is a **weak order**.
Representability

**Definition**
A scoring function $f$ represents a preference relation $\succ$ if for all $x, y$

$$x \succ y \iff f(x) > f(y).$$

**Necessary condition for representability**
The preference relation $\succ$ is a weak order.

**Sufficient condition for representability**
- $\succ$ is a weak order
- the domain is countable or some continuity conditions are satisfied (studied in decision theory)
Not every WO can be represented using a scoring function.
Not every WO can be represented using a scoring function

**Lexicographic order in \( R \times R \)**

\[(x_1, y_1) \succ^\text{lo} (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)\]
Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

Proof

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^lo (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_0, 0) > f(x_0, 1)$.
6. So there are uncountably many nonempty disjoint intervals in $R$.
7. Each such interval contains a rational number: contradiction with the countability of the set of rational numbers.
Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2)$$

Proof

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
Not every WO can be represented using a scoring function

**Lexicographic order in** $R \times R$

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

**Proof**

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^lo (x_0, 0)$. 
Not every WO can be represented using a scoring function

**Lexicographic order in $R \times R$**

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

**Proof**

1. Assume there is a real-valued function $f$ such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
Not every WO can be represented using a scoring function

Lexicographic order in $R \times R$

$$(x_1, y_1) \succ^{lo} (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

Proof

1. Assume there is a real-valued function $f$ such that $x \succ^{lo} y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^{lo} (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$. 

Each such interval contains a rational number: contradiction with the countability of the set of rational numbers.
Lexicographic order in $\mathbb{R} \times \mathbb{R}$

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

Proof

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^lo (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$. 

Not every WO can be represented using a scoring function
Not every WO can be represented using a scoring function.

**Lexicographic order in $R \times R$**

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

**Proof**

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^lo (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$.
6. So there are uncountably many nonempty disjoint intervals in $R$. 

Jan Chomicki () Preference Queries 16 / 65
Not every WO can be represented using a scoring function

**Lexicographic order in** $R \times R$

$$(x_1, y_1) \succ^lo (x_2, y_2) \equiv x_1 > x_2 \lor (x_1 = x_2 \land y_1 > y_2)$$

**Proof**

1. Assume there is a real-valued function $f$ such that $x \succ^lo y \equiv f(x) > f(y)$.
2. For every $x_0$, $(x_0, 1) \succ^lo (x_0, 0)$.
3. Thus $f(x_0, 1) > f(x_0, 0)$.
4. Consider now $x_1 > x_0$.
5. Clearly $f(x_1, 1) > f(x_1, 0) > f(x_0, 1) > f(x_0, 0)$.
6. So there are uncountably many nonempty disjoint intervals in $R$.
7. Each such interval contains a rational number: contradiction with the countability of the set of rational numbers.
Preference constructors [Kie02, KK02]

Good values
Prefer \( v \in S_1 \) over \( v \notin S_1 \).

POS(Make, \{mazda, vw\})

Bad values
Prefer \( v \notin S_1 \) over \( v \in S_1 \).

NEG(Make, \{yugo\})

Explicit preference
Preference encoded by a finite directed graph.

EXP(Make, \{(bmw, ford), ..., (mazda, kia)\})

Value comparison
Prefer larger/smaller values.

HIGHEST(Year)
LOWEST(Price)

Distance
Prefer values closer to \( v_0 \).

AROUND(Price, 12K)
Good values

Prefer $v \in S_1$ over $v \notin S_1$. 
Preference constructors [Kie02, KK02]

**Good values**

Prefer $v \in S_1$ over $v \not\in S_1$.

**POS(Make, \{mazda, vw\})**

**Bad values**

Prefer $v \not\in S_1$ over $v \in S_1$.

**NEG(Make, \{yugo\})**

**Explicit preference**

Preference encoded by a finite directed graph.

**EXP(Make, \{(bmw, ford), \ldots, (mazda, kia)\})**

**Value comparison**

Prefer larger/smaller values.

**HIGHEST(Year)**

**LOWEST(Price)**

**Distance**

Prefer values closer to $v_0$.

**AROUND(Price, 12K)**
Preference constructors [Kie02, KK02]

**Good values**
- Prefer $v \in S_1$ over $v \notin S_1$.

**Bad values**
- Prefer $v \notin S_1$ over $v \in S_1$.

$$\text{POS}(\text{Make}, \{\text{mazda, vw}\})$$

Explicit preference
- Preference encoded by a finite directed graph.

$$\text{EXP}(\text{Make}, \{(\text{bmw, ford}), \ldots, (\text{mazda, kia})\})$$

Value comparison
- Prefer larger/smaller values.

$$\text{HIGHEST}(\text{Year})$$

$$\text{LOWEST}(\text{Price})$$

Distance
- Prefer values closer to $v_0$.

$$\text{AROUND}(\text{Price}, 12K)$$
### Preference constructors [Kie02, KK02]

<table>
<thead>
<tr>
<th>Good values</th>
<th>POS(Make, {mazda, vw})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer $v \in S_1$</td>
<td></td>
</tr>
<tr>
<td>over $v \notin S_1$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bad values</th>
<th>NEG(Make, {yugo})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer $v \notin S_1$</td>
<td></td>
</tr>
<tr>
<td>over $v \in S_1$</td>
<td></td>
</tr>
</tbody>
</table>
Preference constructors [Kie02, KK02]

Good values
Prefer $v \in S_1$ over $v \notin S_1$.

Bad values
Prefer $v \notin S_1$ over $v \in S_1$.

Explicit preference
Preference encoded by a finite directed graph.

POS(Make, \{mazda, vw\})

NEG(Make, \{yugo\})
Preference constructors [Kie02, KK02]

Good values
Prefer \( v \in S_1 \) over \( v \not\in S_1 \).

Bad values
Prefer \( v \not\in S_1 \) over \( v \in S_1 \).

Explicit preference
Preference encoded by a finite directed graph.

POS(Make,\{mazda,vw\})
NEG(Make,\{yugo\})
EXP(Make,\{(bmw,ford),..., (mazda,kia)\})
Preference constructors [Kie02, KK02]

Good values
Prefer $v \in S_1$ over $v \notin S_1$.

Bad values
Prefer $v \notin S_1$ over $v \in S_1$.

Explicit preference
Preference encoded by a finite directed graph.

Value comparison
Prefer larger/smaller values.

POS(Make,\{mazda,vw\})
NEG(Make,\{yugo\})
EXP(Make,\{(bmw,ford),..., (mazda,kia)\})
### Preference constructors [Kie02, KK02]

<table>
<thead>
<tr>
<th>Good values</th>
<th>POS(Make, {mazda, vw})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer ( v \in S_1 ) over ( v \notin S_1 ).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bad values</th>
<th>NEG(Make, {yugo})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer ( v \notin S_1 ) over ( v \in S_1 ).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Explicit preference</th>
<th>EXP(Make, {(bmw, ford), ..., (mazda, kia)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preference encoded by a finite directed graph.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value comparison</th>
<th>HIGHEST(Year) LOWEST(Price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer larger/smaller values.</td>
<td></td>
</tr>
<tr>
<td>Good values</td>
<td>Prefer $v \in S_1$ over $v \notin S_1$.</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>POS(Make, {mazda, vw})</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bad values</th>
<th>Prefer $v \notin S_1$ over $v \in S_1$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG(Make, {yugo})</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Explicit preference</th>
<th>Preference encoded by a finite directed graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP(Make, {(bmw, ford), ..., (mazda, kia)})</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value comparison</th>
<th>Prefer larger/smaller values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHEST(Year)</td>
<td>LOWEST(Price)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distance</th>
<th>Prefer values closer to $v_0$.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Preference constructors [Kie02, KK02]

Good values
Prefer \( v \in S_1 \) over \( v \notin S_1 \).

POS(Make, \{mazda, vw\})

Bad values
Prefer \( v \notin S_1 \) over \( v \in S_1 \).

NEG(Make, \{yugo\})

Explicit preference
Preference encoded by a finite directed graph.

EXP(Make, \{(bmw, ford), ..., (mazda, kia)\})

Value comparison
Prefer larger/smaller values.

HIGHEST(Year)
LOWEST(Price)

Distance
Prefer values closer to \( v_0 \).

AROUND(Price, 12K)
Combining preferences

Preference composition: combining preferences about objects of the same kind, dimensionality is not increased.

Preference accumulation: defining preferences over objects in terms of preferences over simpler objects, dimensionality is increased (preferences over Cartesian product).
Combining preferences

Preference composition
- combining preferences about objects of the same kind
- dimensionality is not increased
- representing preference aggregation, revision, ...
Combining preferences

Preference composition

- combining preferences about objects of the same kind
- dimensionality is not increased
- representing preference aggregation, revision, ...

Preference accumulation

- defining preferences over objects in terms of preferences over simpler objects
- dimensionality is increased (preferences over Cartesian product).
Combining preferences: composition

Boolean composition

\[ x \succ \bigcup y \equiv x \succ 1 y \lor x \succ 2 y \]

and similarly for \( \cap \).

Prioritized composition

\[ x \succ_{\text{lex}} y \equiv x \succ 1 y \lor (y \nmid 1 x \land x \succ 2 y) \]

Pareto composition

\[ x \succ_{\text{Par}} y \equiv (x \succ 1 y \land y \nmid 2 x) \lor (x \succ 2 y \land y \nmid 1 x) \]
Combining preferences: composition

Boolean composition

\[ x \succ^U y \equiv x \succ_1 y \lor x \succ_2 y \]

and similarly for \( \cap \).
Combining preferences: composition

Boolean composition

\[ x \succeq^\cup y \equiv x \succeq_1 y \lor x \succeq_2 y \]

and similarly for \( \cap \).

Prioritized composition

\[ x \succeq^{lex} y \equiv x \succeq_1 y \lor (y \not\succeq_1 x \land x \succeq_2 y). \]
Combining preferences: composition

Boolean composition

\[ x \succ^\cup y \equiv x \succ_1 y \lor x \succ_2 y \]

and similarly for \( \cap \).

Prioritized composition

\[ x \succ^{\text{lex}} y \equiv x \succ_1 y \lor (y \not\succ_1 x \land x \succ_2 y) \].

Pareto composition

\[ x \succ^{\text{Par}} y \equiv (x \succ_1 y \land y \not\succ_2 x) \lor (x \succ_2 y \land y \not\succ_1 x) \].
Preference composition

Preference relation $\succ_1$
- bmw
- ford
- mazda
- kia

Preference relation $\succ_2$
- bmw
- ford
- mazda
- kia

Prioritized composition
- bmw
- ford
- mazda
- kia

Pareto composition
- bmw
- ford
- mazda
- kia
Preference composition

Preference relation $\succ_1$

bmw

ford  mazda

kia
Preference composition

Preference relation $\succ_1$

- BMW
- Ford
- Mazda
- Kia

Preference relation $\succ_2$

- Ford
- Mazda
- Kia
- BMW

Jan Chomicki ()
Preference Queries
Preference composition

Preference relation $\succ_1$

- bmw
- ford
- mazda
- kia

Preference relation $\succ_2$

- ford
- mazda
- kia
- bmw

Prioritized composition

- bmw
  - ford
    - mazda
      - kia
Preference composition

Preference relation $\succ_1$

- bmw
- ford
- mazda
- kia

Preference relation $\succ_2$

- ford
- kia
- mazda
- bmw

Prioritized composition

- bmw
- ford
- mazda
- kia

Pareto composition

- ford
- bmw
- kia
- mazda
Combining preferences: accumulation [Kie02]

Prioritized accumulation:
\[ \succ_{pr} = (\succ_1 \& \succ_2) (x_1, x_2) \succ_{pr} (y_1, y_2) \equiv x_1 \succ_1 y_1 \lor (x_1 = y_1 \land x_2 \succ_2 y_2) \]

Pareto accumulation:
\[ \succ_{pa} = (\succ_1 \otimes \succ_2) (x_1, x_2) \succ_{pa} (y_1, y_2) \equiv (x_1 \succ_1 y_1 \land x_2 \succeq_2 y_2) \lor (x_1 \succeq_1 y_1 \land x_2 \succ_2 y_2) \]

Properties:
closure, associativity, commutativity of Pareto accumulation
Prioritized accumulation: $\succeq_{pr} = (\succeq_1 \& \succeq_2)$

$$(x_1, x_2) \succeq_{pr} (y_1, y_2) \equiv x_1 \succeq_1 y_1 \lor (x_1 = y_1 \land x_2 \succeq_2 y_2).$$
Combining preferences: accumulation [Kie02]

Prioritized accumulation: $\succ^{pr} = (\succ_1 \& \succ_2)$

$$(x_1, x_2) \succ^{pr} (y_1, y_2) \equiv x_1 \succ_1 y_1 \lor (x_1 = y_1 \land x_2 \succ_2 y_2).$$

Pareto accumulation: $\succ^{pa} = (\succ_1 \otimes \succ_2)$

$$(x_1, x_2) \succ^{pa} (y_1, y_2) \equiv (x_1 \succ_1 y_1 \land x_2 \succeq_2 y_2) \lor (x_1 \succeq_1 y_1 \land x_2 \succ_2 y_2).$$
Combining preferences: accumulation [Kie02]

**Prioritized accumulation:** \( \succeq^{pr} = (\succeq_1 \& \succeq_2) \)

\[
(x_1, x_2) \succeq^{pr} (y_1, y_2) \equiv x_1 \succeq_1 y_1 \lor (x_1 = y_1 \land x_2 \succeq_2 y_2).
\]

**Pareto accumulation:** \( \succeq^{pa} = (\succeq_1 \otimes \succeq_2) \)

\[
(x_1, x_2) \succeq^{pa} (y_1, y_2) \equiv (x_1 \succeq_1 y_1 \land x_2 \succeq_2 y_2) \lor (x_1 \succeq_1 y_1 \land x_2 \succeq_2 y_2).
\]

**Properties**

- closure
- associativity
- commutativity of Pareto accumulation
Given single-attribute total preference relations $\succ A_1, \ldots, \succ A_n$ for a relational schema $R(A_1, \ldots, A_n)$, the skyline preference relation $\succ_{sky}$ is defined as:

$$\succ_{sky} = \succ A_1 \otimes \succ A_2 \otimes \cdots \otimes \succ A_n.$$  

Unfolding the definition for $$(x_1, \ldots, x_n) \succ_{sky} (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \succeq A_i y_i \land \bigvee_i x_i \succ A_i y_i.$$  

In two-dimensional Euclidean space $$(x_1, x_2) \succ_{sky} (y_1, y_2) \equiv x_1 \geq y_1 \land x_2 > y_2 \lor x_1 > y_1 \land x_2 \geq y_2.$$
Given single-attribute total preference relations $\succ A_1, \ldots, \succ A_n$ for a relational schema $R(A_1, \ldots, A_n)$, the *skyline* preference relation $\succ^{sky}$ is defined as

\[
\succ^{sky} \equiv \succ A_1 \otimes \succ A_2 \otimes \cdots \otimes \succ A_n .
\]

Unfolding the definition

\[
(x_1, \ldots, x_n) \succ^{sky} (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \succeq A_i y_i \land \bigvee_i x_i \succ A_i y_i .
\]
Skyline

Given single-attribute total preference relations \( \succ A_1, \ldots, \succ A_n \) for a relational schema \( R(A_1, \ldots, A_n) \), the 

**skyline** preference relation \( \succ^{sky} \) is defined as

\[
\succ^{sky} = \succ A_1 \otimes \succ A_2 \otimes \cdots \otimes \succ A_n.
\]

Unfolding the definition

\[
(x_1, \ldots, x_n) \succ^{sky} (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \succeq_{A_i} y_i \land \bigvee_i x_i \succ_{A_i} y_i.
\]

Two-dimensional Euclidean space

\[
(x_1, x_2) \succ^{sky} (y_1, y_2) \equiv x_1 \geq y_1 \land x_2 > y_2 \lor x_1 > y_1 \land x_2 \geq y_2
\]
Skyline in Euclidean space
Skyline in Euclidean space

Maximal skyline vectors

A skyline consists of all maxima of monotonic scoring functions.

Skyline is not a WO (\(2, 0\)) \(#\) \((0, 2)\) \(#\) \((1, 0)\), \((2, 0)\) \(#\) \((1, 0)\).
Skyline in Euclidean space

Maximal skyline vectors

A skyline consists of all maxima of monotonic scoring functions.

Skyline is not a WO (2, 0) ⊁ sky (0, 2), (0, 2) ⊁ sky (1, 0), (2, 0) ≻ sky (1, 0).
Skyline in Euclidean space

Maximal skyline vectors

Maxima

A skyline consists of all maxima of **monotonic** scoring functions.
Skyline in Euclidean space

Maximal skyline vectors

A skyline consists of all maxima of monotonic scoring functions.

Skyline is not a WO

$$(2, 0) \not\succ_{\text{sky}} (0, 2), (0, 2) \not\succ_{\text{sky}} (1, 0), (2, 0) \succ_{\text{sky}} (1, 0)$$
Skyline variants

Groupwise skyline compare only tuples in the same group

Order properties Attribute orders are general SPOs.

Non-Euclidean spaces Metric spaces: distance vectors in road networks

Dynamic attributes Attribute values can change dynamically: distance from query point in road networks
Skyline variants

Groupwise skyline
- compare only tuples in the same group
Skyline variants

Groupwise skyline
- compare only tuples in the same group

Order properties
Attribute orders are general SPOs.
Skyline variants

Groupwise skyline
- compare only tuples in the same group

Order properties
Attribute orders are general SPOs.

Non-Euclidean spaces
Metric spaces:
- distance vectors in road networks
### Skyline variants

<table>
<thead>
<tr>
<th>Groupwise skyline</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare only tuples in the same group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute orders are general SPOs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Euclidean spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric spaces:</td>
</tr>
<tr>
<td>distance vectors in road networks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute values can change dynamically:</td>
</tr>
<tr>
<td>distance from query point in road networks</td>
</tr>
</tbody>
</table>
Combining scoring functions

Scoring functions can be combined using numerical operators.

Common scenario: scoring functions $f_1, \ldots, f_n$ aggregate scoring function:

$$F(t) = E(f_1(t), \ldots, f_n(t))$$

Linear scoring function:

$$\sum_{i=1}^{n} \alpha_i f_i$$

Numerical vs. logical combination:

- Logical combination cannot be defined numerically.
- Numerical combination cannot be defined logically (unless arithmetic operators are available).
Scoring functions can be combined using **numerical** operators.
Combining scoring functions

Scoring functions can be combined using **numerical** operators.

**Common scenario**
- scoring functions $f_1, \ldots, f_n$
- aggregate scoring function: $F(t) = E(f_1(t), \ldots, f_n(t))$
- linear scoring function: $\sum_{i=1}^{n} \alpha_i f_i$
Combining scoring functions

Scoring functions can be combined using **numerical** operators.

**Common scenario**
- scoring functions \( f_1, \ldots, f_n \)
- aggregate scoring function: \( F(t) = E(f_1(t), \ldots, f_n(t)) \)
- linear scoring function: \( \sum_{i=1}^{n} \alpha_i f_i \)

**Numerical vs. logical combination**
- logical combination cannot be defined numerically
- numerical combination cannot be defined logically (unless arithmetic operators are available)
Part II

Preference Queries
Outline of Part II

2 Preference queries
- Retrieving non-dominated elements
- Rewriting queries with winnow
- Retrieving Top-$K$ elements
- Optimizing Top-$K$ queries
Winnow[Cho03]

new relational algebra operator \( \omega \) (other names: Best, BMO [Kie02]) retrieves the non-dominated (best) elements in a database relation can be expressed in terms of other operators

**Definition**

Given a preference relation \( \succ \) and a database relation \( r \):

\[
\omega \succ (r) = \{ t \in r | \neg \exists t' \in r. t' \succ t \}.
\]

Notation: If a preference relation \( \succ_C \) is defined using a formula \( C \), then we write \( \omega_C(r) \), instead of \( \omega \succ_C (r) \).

Skyline query \( \omega \succ_{\text{sky}} (r) \) computes the set of maximal vectors in \( r \) (the skyline set).
Winnow

- new relational algebra operator $\omega$ (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

**Definition**

Given a preference relation $\succ$ and a database relation $r$:

$$\omega \succ (r) = \{ t \in r | \neg \exists t' \in r. t' \succ t \}.$$ 

**Notation:** If a preference relation $\succ_C$ is defined using a formula $C$, then we write $\omega_C(r)$, instead of $\omega \succ_C (r)$. 

**Skyline query**

$\omega \succ \text{sky}(r)$ computes the set of maximal vectors in $r$ (the skyline set).
Winnow[Cho03]

- new relational algebra operator $\omega$ (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

**Definition**

Given a preference relation $\succ$ and a database relation $r$:

$$\omega_\succ(r) = \{ t \in r \mid \neg \exists t' \in r. t' \succ t \}.$$
Winnow

- new relational algebra operator \( \omega \) (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

**Definition**

Given a preference relation \( \succ \) and a database relation \( r \):

\[
\omega_{\succ}(r) = \{ t \in r \mid \neg \exists t' \in r. t' \succ t \}.
\]

Notation: If a preference relation \( \succ_C \) is defined using a formula \( C \), then we write \( \omega_C(r) \), instead of \( \omega_{\succ_C}(r) \).
Winnow

- new relational algebra operator \( \omega \) (other names: Best, BMO [Kie02])
- retrieves the non-dominated (best) elements in a database relation
- can be expressed in terms of other operators

**Definition**

Given a preference relation \( \succ \) and a database relation \( r \):

\[
\omega_{\succ}(r) = \{ t \in r \mid \neg \exists t' \in r. t' \succ t \}.
\]

Notation: If a preference relation \( \succ_C \) is defined using a formula \( C \), then we write \( \omega_C(r) \), instead of \( \omega_{\succ_C}(r) \).

**Skyline query**

\( \omega_{\succ_{\text{sky}}}(r) \) computes the set of maximal vectors in \( r \) (the skyline set).
Example of winnow

Relation Car (Make, Year, Price)

Preference relation: (m, y, p) ≻ \(m', y', p'\) ≡ y > y' ∨ (y = y' ∧ p < p')

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20K</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15K</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12K</td>
</tr>
</tbody>
</table>
Example of winnow

Relation \( \text{Car}(\text{Make}, \text{Year}, \text{Price}) \)

Preference relation:

\[
(m, y, p) \succ_1 (m', y', p') \iff y > y' \lor (y = y' \land p < p').
\]
Example of winnow

Relation $\text{Car}(\text{Make}, \text{Year}, \text{Price})$

Preference relation:

$$(m, y, p) \succeq_1 (m', y', p') \equiv y > y' \lor (y = y' \land p < p').$$

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20K</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15K</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12K</td>
</tr>
</tbody>
</table>
Example of winnow

Relation \( \text{Car}(\text{Make}, \text{Year}, \text{Price}) \)

Preference relation:

\[(m, y, p) \succ_1 (m', y', p') \equiv y > y' \lor (y = y' \land p < p').\]

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20K</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15K</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12K</td>
</tr>
</tbody>
</table>
Computing winnow using BNL [BKS01]

Require: \( SPO \succ, \) database relation \( r \)

1: initialize window \( W \) and temporary file \( F \) to empty
2: repeat
3:   for every tuple \( t \) in the input do
4:     if \( t \) is dominated by a tuple in \( W \) then
5:       ignore \( t \)
6:     else if \( t \) dominates some tuples in \( W \) then
7:       eliminate them and insert \( t \) into \( W \)
8:     else if there is room in \( W \) then
9:       insert \( t \) into \( W \)
10:   else
11:     add \( t \) to \( F \)
12:   end if
13: end for
14: output tuples from \( W \) that were added when \( F \) was empty
15: make \( F \) the input, clear \( F \)
16: until empty input
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 
BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Temporary file

Window

Input

c,e,d,a,b
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.
BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.
BNL in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$.

Window

Input

Temporary file

Jan Chomicki
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 

Temporary file

Window

Input

Jan Chomicki ()
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 

Temporary file

```
Input
```

```
Window
```

```

Jan Chomicki () Preference Queries 31 / 65
```
Preference relation: $a \triangleright c$, $a \triangleright d$, $b \triangleright e$. 

Temporary file

Window

Input

Jan Chomicki
BNL in action

Preference relation: \( a \succ c, a \succ d, b \succ e. \)

Temporary file

Window

\[
\begin{array}{c}
a \\
b
\end{array}
\]

Input
Computing winnow with presorting

SFS: adding presorting step to BNL [CGGL03]

- topologically sort the input:
  - if \( x \) dominates \( y \), then \( x \) precedes \( y \) in the sorted input

- window contains only winnow points and can be output after every pass

- for skylines: sort the input using a monotonic scoring function, for example \( \prod_{i=1}^{k} x_i \).

LESS: integrating different techniques [GSG07]

- adding an elimination filter to the first external sort pass
- combining the last external sort pass with the first SFS pass

- average running time: \( O(kn) \)
Computing winnow with presorting

SFS: adding presorting step to BNL [CGGL03]

- topologically sort the input:
  - if \( x \) dominates \( y \), then \( x \) precedes \( y \) in the sorted input
  - window contains only winnow points and can be output after every pass
- for skylines: sort the input using a monotonic scoring function, for example \( \prod_{i=1}^{k} x_i \).
Computing winnow with presorting

**SFS: adding presorting step to BNL [CGGL03]**

- topologically sort the input:
  - if \( x \) dominates \( y \), then \( x \) precedes \( y \) in the sorted input
  - window contains only winnow points and can be output after every pass
- for skylines: sort the input using a monotonic scoring function, for example \( \prod_{i=1}^{k} x_i \).

**LESS: integrating different techniques [GSG07]**

- adding an elimination filter to the first external sort pass
- combining the last external sort pass with the first SFS pass
- average running time: \( O(kn) \)
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 

Temporary file

Window

Input

a,b,c,d,e
SFS in action

Preference relation: $a \succ c, a \succ d, b \succ e$.
Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 

Window

Temporary file

Input
c, d, e
SFS in action

Preference relation: $a \succ c$, $a \succ d$, $b \succ e$. 

Temporary file

Window

Input

d,e
SFS in action

Preference relation: \( a \succ c, a \succ d, b \succ e. \)

Window

<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

Temporary file

Input

| e |
SFS in action

Preference relation: \( a \succ c, a \succ d, b \succ e. \)
Generalizations of winnow

Iterating winnow

\[ \omega^{n+1} \succ (r) = \omega \succ (r - \bigcup_{1 \leq i \leq n} \omega_i \succ (r)) \]

Ranking tuples by their minimum distance from a winnow tuple:

\[ \eta \succ (r) = \{ (t, i) | t \in \omega_i \cap (r) \} \]

Return the tuples dominated by at most \( k \) tuples:

\[ \omega \succ (r) = \{ t \in r | \# \{ t' \in r | t' \succ t \} \leq k \} \]
Generalizations of winnow

Iterating winnow

\[ \omega^0_\succ (r) = \omega_\succ (r) \]

\[ \omega^{n+1}_\succ (r) = \omega_\succ (r - \bigcup_{1 \leq i \leq n} \omega^i_\succ (r)) \]
Generalizations of winnow

Iterating winnow

\[ \omega_\succ^0 (r) = \omega_\succ (r) \]
\[ \omega_\succ^{n+1} (r) = \omega_\succ (r - \bigcup_{1 \leq i \leq n} \omega_i^\succ (r)) \]

Ranking

Rank tuples by their minimum distance from a winnow tuple:

\[ \eta_\succ (r) = \{(t, i) \mid t \in \omega_C^i (r)\} \]
Generalizations of winnow

Iterating winnow

\[ \omega_0^0 (r) = \omega_\succ (r) \]
\[ \omega_\succ^{n+1} (r) = \omega_\succ (r - \bigcup_{1 \leq i \leq n} \omega_i^i (r)) \]

Ranking

Rank tuples by their minimum distance from a winnow tuple:

\[ \eta_\succ (r) = \{ (t, i) \mid t \in \omega_i^i (r) \} . \]

k-band

Return the tuples dominated by at most \( k \) tuples:

\[ \omega_\succ (r) = \{ t \in r \mid \# \{ t' \in r \mid t' \succ t \} \leq k \} . \]
Preference SQL

The language

basic preference constructors

Pareto/prioritized accumulation

new SQL clause

PREFERRING
groupwise preferences

implementation: translation to SQL

Winnow in Preference SQL

SELECT * FROM Car

PREFERRING HIGHEST(Year)

CASCADE LOWEST(Price)
The language

- basic preference constructors
- Pareto/prioritized accumulation
- new SQL clause PREFERING
- groupwise preferences
- implementation: translation to SQL
Preference SQL

The language
- basic preference constructors
- Pareto/prioritized accumulation
- new SQL clause PREFERING
- groupwise preferences
- implementation: translation to SQL

Winnow in Preference SQL

```
SELECT * FROM Car
PREFERING HIGHEST(Year)
   CASCADE LOWEST(Price)
```
Commutativity of winnow with selection

If the formula

\[ \forall t_1, t_2. \left[ \alpha(t_2) \land \gamma(t_1, t_2) \right] \Rightarrow \alpha(t_1) \]

is valid, then for every

\[ \sigma \alpha(\omega \gamma(r)) = \omega \gamma(\sigma \alpha(r)) \]

Under the preference relation \((m, y, p) \succ C_1(m', y', p') \equiv y > y' \land p \leq p' \lor y \geq y' \land p < p'\)

the selection \(\sigma \) Price < 20 does not commute with \(\omega C_1\).
Commutativity of winnow with selection

If the formula

$$\forall t_1, t_2. [\alpha(t_2) \land \gamma(t_1, t_2)] \Rightarrow \alpha(t_1)$$

is valid, then for every $r$

$$\sigma_\alpha(\omega_\gamma(r)) = \omega_\gamma(\sigma_\alpha(r)).$$
Commutativity of winnow with selection

If the formula

$$\forall t_1, t_2. [\alpha(t_2) \land \gamma(t_1, t_2)] \Rightarrow \alpha(t_1)$$

is valid, then for every $$r$$

$$\sigma_{\alpha}(\omega_\gamma(r)) = \omega_\gamma(\sigma_{\alpha}(r)).$$

Under the preference relation

$$(m, y, p) \succ c_1 (m', y', p') \equiv y > y' \land p \leq p' \lor y \geq y' \land p < p'$$

the selection $$\sigma_{\text{Price}<20K}$$ commutes with $$\omega_{c_1}$$ but $$\sigma_{\text{Price}>20K}$$ does not.
Other algebraic laws

Distributivity of winnow over Cartesian product
For every $r_1$ and $r_2$:

$$\omega C(r_1 \times r_2) = \omega C(r_1) \times r_2$$

if $C$ refers only to the attributes of $r_1$.

Commutativity of winnow
If $\forall t_1, t_2. [C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ is valid and $\succ C_1$ and $\succ C_2$ are SPOs, then for all finite instances $r$:

$$\omega C_1(\omega C_2(r)) = \omega C_2(\omega C_1(r)) = \omega C_2(r)$$
Other algebraic laws

**Distributivity of winnow over Cartesian product**

For every \( r_1 \) and \( r_2 \)

\[
\omega_C(r_1 \times r_2) = \omega_C(r_1) \times r_2
\]

if \( C \) refers only to the attributes of \( r_1 \).

**Commutativity of winnow**

If \( \forall t_1, t_2. [C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)] \) is valid and \( \succ_C^1 \) and \( \succ_C^2 \) are SPOs, then for all finite instances \( r \):

\[
\omega_{C_1}(\omega_{C_2}(r)) = \omega_{C_2}(\omega_{C_1}(r)) = \omega_{C_2}(r).
\]
Semantic query optimization [Cho07b]

Using information about integrity constraints to:
- eliminate redundant occurrences of winnow.
- make more efficient computation of winnow possible.

Eliminating redundancy

Given a set of integrity constraints $F$, $\omega$ is redundant w.r.t. $F$ iff $F$ implies the formula:

$$\forall t_1, t_2. R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2.$$
Using information about integrity constraints to:
- eliminate redundant occurrences of winnow.
- make more efficient computation of winnow possible.

Eliminating redundancy

Given a set of integrity constraints \( F \), \( \omega_C \) is redundant w.r.t. \( F \) iff \( F \) implies the formula

\[
\forall t_1, t_2. \ R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2.
\]
Constraint-generating dependencies (CGD) \cite{BCW99, ZO97}

\[ \forall t_1 \ldots \forall t_n. [R(t_1) \land \ldots \land R(t_n) \land \gamma(t_1, \ldots, t_n) \Rightarrow \gamma'(t_1, \ldots, t_n).] \]
Integrity constraints

Constraint-generating dependencies (CGD) [BCW99, ZO97]

\[ \forall t_1 \ldots \forall t_n. [R(t_1) \land \cdots \land R(t_n) \land \gamma(t_1, \ldots t_n)] \Rightarrow \gamma'(t_1, \ldots t_n). \]
Constraint-generating dependencies (CGD) [BCW99, ZO97]

\[ \forall t_1 \ldots \forall t_n. [R(t_1) \land \cdots \land R(t_n) \land \gamma(t_1, \ldots t_n)] \Rightarrow \gamma'(t_1, \ldots t_n). \]

CGD entailment

Decidable by reduction to the validity of \( \forall \)-formulas in the constraint theory (assuming the theory is decidable).
Top-\(K\) queries

Each tuple \(t\) in a relation has numeric scores \(f_1(t), \ldots, f_m(t)\) assigned by numeric component scoring functions \(f_1, \ldots, f_m\). The aggregate score of \(t\) is \(F(t) = E(f_1(t), \ldots, f_m(t))\) where \(E\) is a numeric-valued expression. \(F\) is monotone if \(E(x_1, \ldots, x_m) \leq E(y_1, \ldots, y_m)\) whenever \(x_i \leq y_i\) for all \(i\).

Top-\(K\) queries return \(K\) elements having top \(F\)-values in a database relation \(R\). The query can be expressed in an extension of SQL:

```
SELECT *
FROM R
ORDER BY F DESC
LIMIT K
```
Top-$K$ queries

Scoring functions

- each tuple $t$ in a relation has numeric scores $f_1(t), \ldots, f_m(t)$ assigned by numeric component scoring functions $f_1, \ldots, f_m$
- the aggregate score of $t$ is $F(t) = E(f_1(t), \ldots, f_m(t))$ where $E$ is a numeric-valued expression
- $F$ is monotone if $E(x_1, \ldots, x_m) \leq E(y_1, \ldots, y_m)$ whenever $x_i \leq y_i$ for all $i$
Top-$K$ queries

Scoring functions

- each tuple $t$ in a relation has numeric scores $f_1(t), \ldots, f_m(t)$ assigned by numeric component scoring functions $f_1, \ldots, f_m$
- the aggregate score of $t$ is $F(t) = E(f_1(t), \ldots, f_m(t))$ where $E$ is a numeric-valued expression
- $F$ is monotone if $E(x_1, \ldots, x_m) \leq E(y_1, \ldots, y_m)$ whenever $x_i \leq y_i$ for all $i$

Top-$K$ queries

- return $K$ elements having top $F$-values in a database relation $R$
- query expressed in an extension of SQL:
  
  ```sql
  SELECT *
  FROM R
  ORDER BY F DESC
  LIMIT K
  ```
Top-\(K\) sets

Definition

Given a scoring function \(F\) and a database relation \(r\), \(s\) is a Top-\(K\) set if:

\[
\forall t \in s \land \forall t' \in r - s. \quad F(t) \geq F(t')
\]

\(|s| = \min(K, |r|)\)

\(s \subseteq r\)

There may be more than one Top-\(K\) set: one is selected non-deterministically.
Top-$K$ sets

**Definition**

Given a scoring function $F$ and a database relation $r$, $s$ is a Top-$K$ set if:

- $s \subseteq r$
- $|s| = \min(K, |r|)$
- $\forall t \in s. \forall t' \in r - s. \; F(t) \geq F(t')$

There may be more than one Top-$K$ set: one is selected non-deterministically.
Example of Top-2

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
<th>Aggregate score</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20000</td>
<td>4000</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15000</td>
<td>9000</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Component scoring functions:

\[
f_1(m, y, p) = (y - 2005) \]

\[
f_2(m, y, p) = (20000 - p) \]

Aggregate scoring function:

\[
F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p) \]
Example of Top-2

Relation $Car(Make, Year, Price)$

- component scoring functions:
  
  $f_1(m, y, p) = (y - 2005)$

  $f_2(m, y, p) = (20000 - p)$

- aggregate scoring function:
  
  $F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p)$
Example of Top-2

Relation \textit{Car}(\textit{Make}, \textit{Year}, \textit{Price})

- component scoring functions:
  \begin{align*}
  f_1(m, y, p) &= (y - 2005) \\
  f_2(m, y, p) &= (20000 - p)
  \end{align*}

- aggregate scoring function:
  \[ F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p) \]

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
<th>Aggregate score</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20000</td>
<td>4000</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15000</td>
<td>9000</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12000</td>
<td>10000</td>
</tr>
</tbody>
</table>
Example of Top-2

Relation \textit{Car}(\textit{Make}, \textit{Year}, \textit{Price})

- component scoring functions:
  \[ f_1(m, y, p) = (y - 2005) \]
  \[ f_2(m, y, p) = (20000 - p) \]

- aggregate scoring function:
  \[ F(m, y, p) = 1000 \cdot f_1(m, y, p) + f_2(m, y, p) \]

<table>
<thead>
<tr>
<th>Make</th>
<th>Year</th>
<th>Price</th>
<th>Aggregate score</th>
</tr>
</thead>
<tbody>
<tr>
<td>mazda</td>
<td>2009</td>
<td>20000</td>
<td>4000</td>
</tr>
<tr>
<td>ford</td>
<td>2009</td>
<td>15000</td>
<td>9000</td>
</tr>
<tr>
<td>ford</td>
<td>2007</td>
<td>12000</td>
<td>10000</td>
</tr>
</tbody>
</table>
Computing Top-$K$

Naive approaches sort, output the first $K$-tuples scan the input maintaining a priority queue of size $K$...

Better approaches the entire input does not need to be scanned... provided additional data structures are available... variants of the threshold algorithm

Jan Chomicki () Preference Queries 43 / 65
Naive approaches

- sort, output the first $K$-tuples
- scan the input maintaining a priority queue of size $K$
- ...
Computing Top-$K$

**Naive approaches**
- sort, output the first $K$-tuples
- scan the input maintaining a priority queue of size $K$
- ...

**Better approaches**
Computing Top-$K$

**Naive approaches**
- sort, output the first $K$-tuples
- scan the input maintaining a priority queue of size $K$
- ...

**Better approaches**
- the entire input does not need to be scanned...
Computing Top-$K$

### Naive approaches
- sort, output the first $K$-tuples
- scan the input maintaining a priority queue of size $K$
- ...

### Better approaches
- the entire input does not need to be scanned...
- ... provided additional data structures are available
Computing Top-$K$

**Naive approaches**
- sort, output the first $K$-tuples
- scan the input maintaining a priority queue of size $K$
- ...

**Better approaches**
- the entire input does not need to be scanned...
- ... provided additional data structures are available
- variants of the threshold algorithm
Threshold algorithm (TA) [FLN03]

Inputs

- A monotone scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- Lists $S_i, i = 1, \ldots, m$, each sorted on $f_i$ (descending) and representing a different ranking of the same set of objects

1. For each list $S_i$ in parallel, retrieve the current object $w$ in sorted order:
   - (Random access) For every $j \neq i$, retrieve $v_j = f_j(w)$ from the list $S_j$
   - If $d = E(v_1, \ldots, v_m)$ is among the highest $K$ scores seen so far, remember $t$ and $d$ (ties broken arbitrarily)

2. Thresholding:
   - For each $i$: If there are already $K$ top-$K$ objects with score at least equal to the threshold $T = E(f_1(w_1), \ldots, f_m(w_m))$, return collected objects sorted by $F$ and terminate
   - Otherwise, go to step 1.
Threshold algorithm (TA)[FLN03]

**Inputs**

- a monotone scoring function \( F(t) = E(f_1(t), \ldots, f_m(t)) \)
- lists \( S_i, \ i = 1, \ldots, m \), each sorted on \( f_i \) (descending) and representing a different ranking of the same set of objects

1. For each list \( S_i \) in parallel retrieve the current object \( w \) in sorted order:
   - (random access) for every \( j \neq i \), retrieve \( v_j = f_j(w) \) from the list \( S_j \)
   - if \( d = E(v_1, \ldots, v_m) \) is among the highest \( K \) scores seen so far, remember \( t \) and \( d \) (ties broken arbitrarily)

2. Thresholding:
   - for each \( i \): \( w_i \) the last object seen under sorted access in \( S_i \)
   - if there are already \( K \) top-\( K \) objects with score at least equal to the threshold \( T = E(f_1(w_1), \ldots, f_m(w_m)) \), return collected objects sorted by \( F \) and terminate
   - otherwise, go to step 1.
**TA in action**

**Aggregate score**

\[ F(t) = P_1(t) + P_2(t) \]

**Priority queue**

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>
TA in action

Aggregate score

\[ F(t) = P_1(t) + P_2(t) \]

Priority queue

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

\( T = 100 \)
**Aggregate score**

\[ F(t) = P_1(t) + P_2(t) \]

**Priority queue**

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

\( T = 100 \)
TA in action

Aggregate score

\[ F(t) = P_1(t) + P_2(t) \]

Priority queue

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

3:80
5:60

\( T = 100 \)
**TA in action**

**Aggregate score**

\[ F(t) = P_1(t) + P_2(t) \]

**Priority queue**

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
<th>|</th>
<th></th>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
<td></td>
<td></td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
<td></td>
<td></td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
<td></td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

\( T = 75 \)

3:80
5:60
TA in action

Aggregate score

\[ F(t) = P_1(t) + P_2(t) \]

Priority queue

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

\( T = 75 \)
**TA in action**

### Aggregate score

\[ F(t) = P_1(t) + P_2(t) \]

### Priority queue

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OID</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ T = 75 \]
TA in databases

- Objects: tuples of a single relation \( r \)
- Single-attribute component scoring functions
- Sorted list access implemented through indexes
- Random access to all lists implemented by primary index access to \( r \) that retrieves entire tuples
• objects: **tuples** of a single relation $r$
• **single-attribute** component scoring functions
• **sorted** list access implemented through **indexes**
• **random** access to all lists implemented by **primary index** access to $r$ that retrieves entire tuples
Optimizing Top-$K$ queries [LCIS05]

Goals
- Integrating Top-$K$ with relational query evaluation and optimization
- Replacing blocking by pipelining

Example

```
SELECT *
FROM Hotel h, Restaurant r, Museum m
WHERE c1 AND c2 AND c3
ORDER BY f1 + f2 + f3
LIMIT K
```
Optimizing Top-$K$ queries [LCIS05]

Goals

- **integrating** Top-$K$ with relational query evaluation and optimization
- replacing blocking by **pipelining**
Optimizing Top-\(K\) queries [LCIS05]

**Goals**
- integrating Top-\(K\) with relational query evaluation and optimization
- replacing blocking by pipelining

**Example**

```sql
SELECT *
FROM Hotel \(h\), Restaurant \(r\), Museum \(m\)
WHERE \(c_1\) AND \(c_2\) AND \(c_3\)
ORDER BY \(f_1 + f_2 + f_3\)
LIMIT \(K\)
```
Optimizing Top-$K$ queries [LCIS05]

**Goals**
- integrating Top-$K$ with relational query evaluation and optimization
- replacing blocking by pipelining

**Example**
```
SELECT *
FROM Hotel $h$, Restaurant $r$, Museum $m$
WHERE $c_1$ AND $c_2$ AND $c_3$
ORDER BY $f_1 + f_2 + f_3$
LIMIT $K$
```

Is there a better evaluation plan than `materialize-then-sort`?
Partial ranking of tuples

Model
set of component scoring functions $P = \{f_1, \ldots, f_m\}$ such that

aggregate scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$

how to rank intermediate tuples?

Ranking principle
Given $P_0 \subseteq P$,

$\bar{F}_{P_0}(t) = E(g_1(t), \ldots, g_m(t))$

where $g_i(t) = \begin{cases} f_i(t) & \text{if } f_i \in P_0 \\ 1 & \text{otherwise} \end{cases}$
Partial ranking of tuples

**Model**

- set of component scoring functions $P = \{f_1, \ldots, f_m\}$ such that $f_i(t) \leq 1$ for all $t$
- aggregate scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- how to rank intermediate tuples?

Jan Chomicki () Preference Queries
Partial ranking of tuples

Model

- set of component scoring functions $P = \{f_1, \ldots, f_m\}$ such that $f_i(t) \leq 1$ for all $t$
- aggregate scoring function $F(t) = E(f_1(t), \ldots, f_m(t))$
- how to rank intermediate tuples?

Ranking principle

Given $P_0 \subseteq P$,

$$\bar{F}_{P_0}(t) = E(g_1(t), \ldots, g_m(t))$$

where

$$g_i(t) = \begin{cases} f_i(t) & \text{if } f_i \in P_0 \\ 1 & \text{otherwise} \end{cases}$$
Relations with rank

Rank-relation $R$

$P_0$ relation $R$

monotone aggregate scoring function $F$

(set of component scoring functions $P_0 \subseteq P$

order:

$t_1 > R_{P_0} t_2 \equiv \bar{F}_{P_0}(t_1) > \bar{F}_{P_0}(t_2)$
## Rank-relation $R_{P_0}$

- relation $R$
- monotone aggregate scoring function $F$ (the same for all relations)
- set of component scoring functions $P_0 \subseteq P$
- order:

\[
t_1 >_{R_{P_0}} t_2 \equiv \bar{F}_{P_0}(t_1) > \bar{F}_{P_0}(t_2)
\]
Ranking intermediate results

Operators

\[ \mu_f (R_{P_0}) \]

\[ t_1 > \mu_f (R_{P_0}) \]

\[ t_2 \equiv \bar{F}_{P_0} \cup \{ f \} (t_1) > \bar{F}_{P_0} \cup \{ f \} (t_2) \]

\[ R_{P_1} \cap S_{P_2} \]

\[ t_1 > R_{P_1} \cap S_{P_2} \]

\[ t_2 \equiv \bar{F}_{P_1} \cup P_{P_2} (t_1) > \bar{F}_{P_1} \cup P_{P_2} (t_2) \]
Operators

- **rank operator** \( \mu_f \): ranks tuples according to an additional component scoring function \( f \)
- **standard relational algebra operators suitably extended** to work on rank-relations
Ranking intermediate results

Operators

- **rank operator** $\mu_f$: ranks tuples according to an additional component scoring function $f$
- standard relational algebra operators suitably extended to work on rank-relations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_f(R_{P_0})$</td>
<td>$t_1 &gt;<em>{\mu_f(R</em>{P_0})} t_2 \equiv \bar{F}<em>{P_0 \cup {f}}(t_1) &gt; \bar{F}</em>{P_0 \cup {f}}(t_2)$</td>
</tr>
<tr>
<td>$R_{P_1} \cap S_{P_2}$</td>
<td>$t_1 &gt;<em>{R</em>{P_1} \cap S_{P_2}} t_2 \equiv \bar{F}<em>{P_1 \cup P_2}(t_1) &gt; \bar{F}</em>{P_1 \cup P_2}(t_2)$</td>
</tr>
</tbody>
</table>
Example

SELECT * FROM S ORDER BY f1 + f2 + f3 LIMIT 1

Unranked relation S

<table>
<thead>
<tr>
<th>A</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Rank-relation S

<table>
<thead>
<tr>
<th>A</th>
<th>f1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.7</td>
</tr>
<tr>
<td>2</td>
<td>2.9</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Jan Chomicki () Preference Queries
Example

Query

```sql
SELECT *
FROM S
ORDER BY f₁ + f₂ + f₃
LIMIT 1
```
Example

Query

```
SELECT * 
FROM S 
ORDER BY f1 + f2 + f3 
LIMIT 1
```

Unranked relation S

<table>
<thead>
<tr>
<th>A</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
</tr>
</tbody>
</table>
Example

Query

```sql
SELECT *
FROM S
ORDER BY f_1 + f_2 + f_3
LIMIT 1
```

Unranked relation $S$

<table>
<thead>
<tr>
<th>A</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Rank-relation $S_{\{f_1\}}$

<table>
<thead>
<tr>
<th>A</th>
<th>$\bar{F}_{{f_1}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.9</td>
</tr>
<tr>
<td>1</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Pipelined execution
### Pipelined execution

<table>
<thead>
<tr>
<th>A</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$\bar{F}_{{f_1}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.9</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
<td>2.5</td>
</tr>
</tbody>
</table>

$\uparrow \text{IndexScan}_{f_1}$
### Pipelined execution

<table>
<thead>
<tr>
<th>A</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$\bar{F}_{{f_1}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.9</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
<td>2.5</td>
</tr>
</tbody>
</table>

$\mu_{f_2}$

$\text{IndexScan}_{f_1}$

<table>
<thead>
<tr>
<th>A</th>
<th>$\bar{F}_{{f_1,f_2}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.75</td>
</tr>
<tr>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>1.95</td>
</tr>
</tbody>
</table>
### Pipelined execution

<table>
<thead>
<tr>
<th>A</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$\bar{F}_{{f_1}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.9</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
<td>2.5</td>
</tr>
</tbody>
</table>

$\mu_{f_2}$

**IndexScan**$_{f_1}$

<table>
<thead>
<tr>
<th>A</th>
<th>$\bar{F}_{{f_1,f_2}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.75</td>
</tr>
<tr>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>1.95</td>
</tr>
</tbody>
</table>

$\mu_{f_3}$

<table>
<thead>
<tr>
<th>A</th>
<th>$\bar{F}_{{f_1,f_2,f_3}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.55</td>
</tr>
<tr>
<td>1</td>
<td>2.4</td>
</tr>
</tbody>
</table>
Algebraic laws for rank-relation operators

Splitting for $\mu_R\{f_1, f_2, \ldots, f_m\} \equiv \mu f_1(\mu f_2(\ldots(\mu f_m(R))\ldots))$

Commutativity of $\mu$

$\mu f_1(\mu f_2(R_P0)) \equiv \mu f_2(\mu f_1(R_P0))$

Commutativity of $\mu$ with selection $\sigma_C(\mu f(R_P0)) \equiv \mu f(\sigma_C(R_P0))$

Distributivity of $\mu$ over Cartesian product $\mu f(R_{P1} \times S_{P2}) \equiv \mu f(R_{P1}) \times S_{P2}$ if $f$ refers only to the attributes of $R$. 
Algebraic laws for rank-relation operators

Splitting for $\mu$

$$R\{f_1, f_2, \ldots, f_m\} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$
Algebraic laws for rank-relation operators

Splitting for $\mu$

$$R_{\{f_1, f_2, \ldots, f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$

Commutativity of $\mu$

$$\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))$$
Algebraic laws for rank-relation operators

**Splitting for \( \mu \)**

\[
R_{\{f_1, f_2, \ldots, f_m\}} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))
\]

**Commutativity of \( \mu \)**

\[
\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))
\]

**Commutativity of \( \mu \) with selection**

\[
\sigma_C(\mu_f(R_{P_0})) \equiv \mu_f(\sigma_C(R_{P_0}))
\]
Algebraic laws for rank-relation operators

Splitting for $\mu$

$$R\{f_1, f_2, \ldots, f_m\} \equiv \mu_{f_1}(\mu_{f_2}(\ldots(\mu_{f_m}(R))\ldots))$$

Commutativity of $\mu$

$$\mu_{f_1}(\mu_{f_2}(R_{P_0})) \equiv \mu_{f_2}(\mu_{f_1}(R_{P_0}))$$

Commutativity of $\mu$ with selection

$$\sigma_C(\mu_f(R_{P_0})) \equiv \mu_f(\sigma_C(R_{P_0}))$$

Distributivity of $\mu$ over Cartesian product

$$\mu_f(R_{P_1} \times S_{P_2}) \equiv \mu_f(R_{P_1}) \times S_{P_2} \text{ if } f \text{ refers only to the attributes of } R.$$
Part III

Preference management
Outline of Part III

3 Preference management
   • Preference modification
Preference modification

Goal

Given a preference relation $\succ$ and additional preference or indifference information $I$, construct a new preference relation $\succ'$ whose contents depend on $\succ$ and $I$.

General postulates fulfillment: the new information $I$ should be completely incorporated into $\succ'$.

Minimal change: $\succ$ should be changed as little as possible.

Closure: order-theoretic properties of $\succ$ should be preserved in $\succ'$.

Finiteness or finite representability of $\succ$ should also be preserved in $\succ'$. 
Preference modification

Goal

Given a preference relation $\succ$ and additional preference or indifference information $I$, construct a new preference relation $\succ'$ whose contents depend on $\succ$ and $I$. 

General postulates

- **fulfillment**: the new information $I$ should be completely incorporated into $\succ'$
- **minimal change**: $\succ$ should be changed as little as possible
- **closure**: order-theoretic properties of $\succ$ should be preserved in $\succ'$
- **finiteness or finite representability**: $\succ$ should also be preserved in $\succ'$
Preference modification

Goal

Given a preference relation $\succ$ and additional preference or indifference information $I$, construct a new preference relation $\succ'$ whose contents depend on $\succ$ and $I$.

General postulates

- **fulfillment**: the new information $I$ should be completely incorporated into $\succ'$
- **minimal change**: $\succ$ should be changed as little as possible
- **closure**:
  - order-theoretic properties of $\succ$ should be preserved in $\succ'$ (SPO, WO)
  - finiteness or finite representability of $\succ$ should also be preserved in $\succ'$
Preference revision [Cho07a]

Setting new information: revising preference relation $\succ_0$ composition operator $\theta$: union, prioritized or Pareto composition

composition eliminates (some) preference conflicts additional assumptions: interval orders $\succ' = TC(\succ_0 \theta \succ_0)$ to guarantee SPO

VW, 2009

Kia, 2009

Kia, 2008

Kia, 2007
Setting

- new information: revising preference relation $\succ_0$
- composition operator $\theta$: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \theta \succ)$ to guarantee SPO
Setting

- new information: revising preference relation $\succ_0$
- composition operator $\theta$: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \theta \succ)$ to guarantee SPO

\[ \begin{align*}
VW, 2009 & \rightarrow VW, 2008 \\
VW, 2008 & \rightarrow VW, 2007 \\
Kia, 2009 & \rightarrow Kia, 2008 \\
Kia, 2008 & \rightarrow Kia, 2007
\end{align*} \]
Preference revision [Cho07a]

**Setting**

- new information: revising preference relation $\succ_0$
- composition operator $\theta$: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \theta \succ)$ to guarantee SPO
Setting

- new information: revising preference relation $\succ_0$
- composition operator $\theta$: union, prioritized or Pareto composition
- composition eliminates (some) preference conflicts
- additional assumptions: interval orders
- $\succ' = TC(\succ_0 \theta \succ)$ to guarantee SPO
Preference contraction [MC08]

Setting new information: contractor relation $\text{CON} \succ \text{CON}': \text{maximal subset of } \succ \text{CONVW}, 2009 \text{VW}, 2008 \text{VW}, 2007 \text{VW}, 2006 \text{VW}, 2005$
Preference contraction [MC08]

Setting

- new information: contractor relation $CON$
- $\succ'$: maximal subset of $\succ$ disjoint with $CON$
Preference contraction [MC08]

**Setting**

- new information: contractor relation $CON$
- $\succ'$: maximal subset of $\succ$ disjoint with $CON$

Preference contraction [MC08]

Setting

- new information: contractor relation $CON$
- $\succ'$: maximal subset of $\succ$ disjoint with $CON$
Preference contraction \([\text{MC08}]\)

**Setting**

- new information: contractor relation \(\text{CON}\)
- \(\succ'\): maximal subset of \(\succ\) disjoint with \(\text{CON}\)
Substitutability [BGS06]
Substitutability [BGS06]

Setting

- new information: set of *indifference* pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving *object substitutability*
Setting

- new information: set of *indifference* pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving *object substitutability*
Substitutability [BGS06]

Setting

- new information: set of indifference pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving object substitutability

VW, 2009
VW, 2008
VW, 2007

Kia, 2009
Kia, 2008
Kia, 2007
Setting

- new information: set of **indifference** pairs
- additional preferences are added to convert indifference to restricted indifference
- achieving **object substitutability**
Part IV

Advanced topics
Prospective research topics

Definability
Given a preference relation \(\succ\), how to construct a definition of a scoring function \(F\) representing \(\succ\), if such a function exists?

Extrinsic preference relations
Preference relations that are not fully defined by tuple contents:
\[x \succ y \equiv \text{BMW}(x) \land \text{Kia}(y)\]
where \(\text{BMW}\) and \(\text{Kia}\) are database relations.

Incomplete preferences
tuple scores and probabilities [SIC08, ZC08]
uncertain tuple scores
disjunctive preferences:
\[a \succ b \lor a \succ c\]
Definability

Given a preference relation $\succ_C$, how to construct a definition of a scoring function $F$ representing $\succ_C$, if such a function exists?
Prospective research topics

Definability

Given a preference relation $\succ_C$, how to construct a **definition** of a scoring function $F$ representing $\succ_C$, if such a function exists?

Extrinsic preference relations

Preference relations that are not fully defined by tuple contents:

$$x \succ y \equiv BMW(x) \land Kia(y)$$

where $BMW$ and $Kia$ are database relations.
Prospective research topics

**Definability**

Given a preference relation $\succeq_C$, how to construct a definition of a scoring function $F$ representing $\succeq_C$, if such a function exists?

**Extrinsic preference relations**

Preference relations that are not fully defined by tuple contents:

$$x \succ y \equiv BMW(x) \land Kia(y)$$

where $BMW$ and $Kia$ are database relations.

**Incomplete preferences**

- tuple scores and probabilities [SIC08, ZC08]
- uncertain tuple scores
- disjunctive preferences: $a \succ b \lor a \succ c$
Preference modification beyond revision and contraction: merging, arbitration,...

general parametric framework?

Conflict resolution

Variations preference and similarity: "find the objects similar to one of the best objects"

Applications preference queries as decision components: workflows, event systems

personalization of query results

preference negotiation: applying contraction
Preference modification

- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution
Preference modification

- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution

Variations

- preference and similarity: “find the objects similar to one of the best objects”
Preference modification
- beyond revision and contraction: merging, arbitration,...
- general parametric framework?
- conflict resolution

Variations
- preference and similarity: “find the objects similar to one of the best objects”

Applications
- preference queries as decision components: workflows, event systems
- personalization of query results
- preference negotiation: applying contraction
Acknowledgments
Denis Mindolin
Sławek Staworko
Xi Zhang
M. Baudinet, J. Chomicki, and P. Wolper.  
Constraint-Generating Dependencies.  
Preliminary version in ICDT'95.

Exploiting Indifference for Customization of Partial Order Skylines.  

S. Börzsönyi, D. Kossmann, and K. Stocker.  
The Skyline Operator.  

J. Chomicki, P. Godfrey, J. Gryz, and D. Liang.  
Skyline with Presorting.  
In *IEEE International Conference on Data Engineering (ICDE)*, 2003.  
Poster.

J. Chomicki.
Preference Formulas in Relational Queries.

J. Chomicki.
Database Querying under Changing Preferences.

J. Chomicki.
Semantic optimization techniques for preference queries.

R. Fagin, A. Lotem, and M. Naor.
Optimal Aggregation Algorithms for Middleware.

P. Godfrey, R. Shipley, and J. Gryz.
Algorithms and Analyses for Maximal Vector Computation.

W. Kießling.
Foundations of Preferences in Database Systems.
In *International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

W. Kießling and G. Köstler.
In *International Conference on Very Large Data Bases (VLDB)*, pages 990–1001, 2002.

C. Li, K. C-C. Chang, I.F. Ilyas, and S. Song.
RankSQL: Query Algebra and Optimization for Relational Top-k Queries.

D. Mindolin and J. Chomicki.
Maximal Contraction of Preference Relations.

M.A. Soliman, I.F. Ilyas, and K. C-C. Chang.
Probabilistic Top-k and Ranking-Aggregate Queries.
X. Zhang and J. Chomicki.
On the Semantics and Evaluation of Top-k Queries in Probabilistic Databases.

X. Zhang and Z. M. Ozsoyoglu.
Implication and Referential Constraints: A New Formal Reasoning.